The temporal correlation of data in a multirate system

Evariste Ntaryamira, Cristian Maxim, Liliana Cucu-Grosjean

The temporal correlation of data in a multirate system

Evariste Ntaryamira
Inria, Paris
evariste.ntaryamira@inria.fr

Cristian Maxim
IRT System X, Saclay
cristian.maxim@irt-systemx.fr

Liliana Cucu-Grosjean
Inria, Paris
liliana.cucu@inria.fr

ABSTRACT

Technologies within embedded real-time systems are continuously evolving making them intelligent; at some point they can achieve targeted functions autonomously. Such systems are extended with capability of sensing the surrounding environment and deciding on their own. In addition to the feasible scheduling policy, the correctness of such decisions highly depends on the quality of the used input data. Thereby, the data management within such systems must fulfill some properties in order to guarantee their correct functioning. In this paper we address the problem of *data temporal correlation and validity* when the system scheduling properties are defined. We present preliminary results on expected properties of the architectures and underline future work.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

scheduling, data, real-time

1. INTRODUCTION AND RELATED WORK

The software embedded in a real-time system is composed of a large number of applications communicating through shared variables. These systems have the capability of making functional decisions autonomously. For instance, the autonomous vehicles are extended with capability of sensing the surrounding environment and navigating on their own by making driving decisions. Additionally to the traditional vehicle functions autonomous vehicles are equipped with algorithms that ease to infer the identity of the objects in the neighborhood. These algorithms work on the data propagating from different paths where these data may be resulting from a same or different source application.

In this paper we consider that the data traversing different paths result from a same source application and only the data produced during a same execution step are meant to be further associated. However, the propagation delays from the source to the associating application may differ from one path to another for different reasons. Hence, waiting queues are used to temporally store the data from the shortest paths until the corresponding data from the longest path arrive. In this paper we propose how to compute the minimum size (later referred to as optimal size) for each of the waiting queues in order to economize the memory resource utilization.

Contribution In this paper we study the temporal correlation of the data propagating along several paths taking into account the timing properties of the system scheduling. We consider the inter-task communication model (circular buffer) offering a predictable data management between communicating applications.

Paper structure. In Section 2 we present the system and the communication models with the associated notations and we formulate the correlation problem. In Section 3 we introduce our first contribution; the data reading strategy easing the correlation process is proposed in 3.1.1, a formal method computing the *Data Consistent Time Interval in 3.1.2*, the formal methods to compute the buffers optimal sizes in 3.1.3 and 3.2.2 and, finally, the proposed data correlation approach is presented in 3.2.1. We conclude and present future work in Section 4.

1.1 Motivating example

An example illustrating the need of associating correlated data is the *FADE* system [8], represented on the Figure 1. *FADE* is a vehicle detection and tracking system composed of a set of image processing components in charge of detecting the characteristics (detection of shadows, headlights, etc.) related to the presence of a vehicle in the neighborhood. For the performance optimization reasons, the image processing is done in a multi-resolution mode. Thereby, only the central part of the image is processed in the high resolution mode (for detecting vehicles being far away) and the periphery of the image is treated in the low resolution mode for detecting closer vehicles.

precisely, the high resolution image initially produced by the IAA[1] is sent to both the CDA[2] and the IPA[3]. Further, the *CDA* application reads the high resolution image as input. The latter is cropped, decimated and processed and, finally, a low resolution image is produced at the *CDA* output port. Further, the low and high resolution images are associated and fused in order to infer the identity of the targeted object.

Obviously, the associated data are shifted by a certain delay induced by the processing of the *CDA*. In order to avoid potential performance degradation, the *IPA* instances must read (from each of the buffers) the data resulting from the same execution step of the IAA [7]. In such a context we say that the used data are *temporally correlated* and, in our paper, we aim to efficiently address the *data temporal*

[1]Image Acquisition Application
[2]Crop and Decimate Application
[3]Image Processing Application

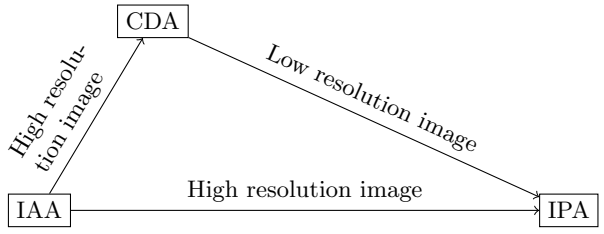

Figure 1: The vehicle detection and tracking system.

correlation problem.

1.2 Related works

The problem related to the association of the data produced during the same execution step (having the same age) has been addressed during the last decades. N. Pontisso and al. in [3, 4] proposed approaches to manage data matching in periodic systems without considering strict properties on the system scheduling; that is, earlier the implementation stage. Their work differs from ours in the sense that the correlation of the data produced during the same execution is ensured taking into account the task system scheduling properties. The FIFO waiting queues are managed sequentially (linearly) while in our work we consider the FIFO buffer to be managed circularly. The circular buffer organization is described in the Section 2.2.

Authors in [7] consider the circular buffer communication model but without considering the task system scheduling properties. In order to ensure the temporal correlation between data from different paths, the communication model is organized as follows: each data sample is double stamped with two different dates; the *timestamp* and *timeOfIssue*. The first, considered as the date of birth of the data sample, is given by the application that initially generated it and is remained unchanged.At the execution completion of each application the data a *timeOfIssue* is added. At the end of the paths, the reader application will retrieve the recent data having the same *timestamp* from the connected buffers In our work we do not time stamp the samples.

2. MODELS AND ANNOTATIONS

In this section, we introduce the system and the communication models as well as the notion of functional chains.

2.1 System Model

We consider a system τ of n periodic tasks $\{\tau_1, \tau_2, \cdots, \tau_n\}$ scheduled preemptively on one processor according to fixed-priority scheduling algorithm. Each task τ_i is described by the tuple (C_i, T_i), where C_i is the worst-case execution time and T_i is the period of the task τ_i. We assume that all tasks are released simultaneously and they have implicit deadline; that is, the deadline is equal to period.

Without any loss of generality, we consider that the tasks are ordered from the highest to the lowest priority and the larger is the task period the lower is the priority. Hence, if $i < j$, then τ_i has a higher priority than τ_j. Each task τ_i generates an infinite number of successive jobs $\tau_{i,j}|j = 1, \cdots, \infty$. We consider that tasks share data through buffers and a task may belong to two different classes: producer or consumer. The shared buffer can be accessed for writing by a single producer while one or multiple consumers can read from it. The data propagation order between tasks does not impact an execution order between those tasks. We describe the data dependencies between the tasks by a graph.

We denote by $G = (V, E)$ such graph, where V is the set of tasks $\{\tau_1, \cdots, \tau_n\}$, E the set of edges and $(\tau_i, \tau_j) \in E$ if τ_j consumes data produced by τ_i.

The graph G may contain different data propagation paths. Hence, we define $\Pi = \{pth_1, \cdots, pth_m\}$ where m is the number paths in Π. A same producer may produce data for several consumers belonging to different paths. For instance, on the Figure 2, the output data of τ_1 is utilized by τ_2 and τ_4 belonging; respectively, to pth_1 and pth_2 where $pth_1 = \{\tau_1, \tau_2, \cdots, \tau_5\}$ and $pth_2 = \{\tau_1, \tau_4, \tau_5\}$. Accordingly, τ_1 is called the *data dispatching task* or simply the *dispatcher* whereas τ_5 is called the *data associating task* or simply the *associator*.

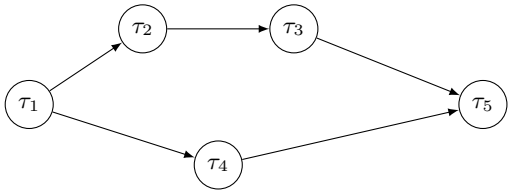

Figure 2: System tasks model

Tasks whose instances consume data produced by the *dispatcher* are referred to as *direct successors* or simply *successors*. Similarly, tasks that produce data for the *associator* are referred to as *direct predecessors* or simply *predecessors*.

2.2 Communication model

Our communication model is the circular buffer. A circular buffer is a *FIFO data structure* that considers memory to be managed circularly; that is, the *read/write* indices loop back to 0 after it reaches the buffer length [1].

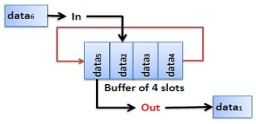

Figure 3: Example of the circular buffer

It has a fixed size allocated once at the system run-time. *tail* and *head* are the pointers indicating the reading and writing positions. Each time a new data sample is inserted into the buffer, the *head* pointer is incremented and likewise, when the data is read the *tail* pointer is incremented. *tail* and *head* are initially set to 0. The modulo operation is performed to reset head or tail to 0, every time the maximum index is reached.

The choice of a circular buffer is motivated by:

1. *The data sample already written in the buffer slot never changes the address until it is overwritten*: it avoids the data shifting from slot to the next one as it it the case for a sequential buffer. Shifting data is a resource consumption which may lead to unpredictable behavior.
2. The utilization of the circular buffer offers a high degree of the communication predictability and no dynamic memory allocation: the required size is computed offline considering the timing characteristics of the communicating tasks.
3. The circular buffer is easy to implement.

From what precedes, we consider a set of buffers $\{\beta_1, \cdots, \beta_x\}$, where each buffer β_x, in addition to *tail* and *head*, is characterized by its cardinality $|\beta_x|$; that is, its size. The size of the buffer gives the information about the number of data samples that can be stored. Each data sample is described by the tuple $\langle dID, dValue \rangle$ where dID is an integer defined as the *data identifier* and $dValue$ is the data sample value.

We consider a single buffer shared between a single producer and one or several consumers.

2.3 Correlation problem formalization

We consider a set of k data paths $\{pth_1, \cdots, pth_k\} \in \Pi$ where the data flowing along the all k paths are initially generated by a same task referred as the ***dispatcher task*** and denoted by τ_{dsp}. These paths are later associated by a same task referred to as the ***associator task*** and denoted by τ_{as}. Accordingly, we call the $\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ and $\{\tau_{as_1}, \cdots, \tau_{as_k}\}$, respectively, the ***successors*** to τ_{dsp} and ***predecessors*** to τ_{as}. For a set of k paths, the data temporal correlation is required if $pth_1 \cap\ pth_2 \cap, \cdots, \cap\ pth_k = \{\tau_{dsp}, \tau_{as}\}$.

Correlation problem formalization:

$$correl(\tau_{as}) = \tau_{disp}\{pth_1(\tau_{dsp_1}, \tau_{as_1}), \cdots, pth_k(\tau_{dsp_k}, \tau_{as_k})\}$$

where k is the number of *paths* involved in the propagation of the data produced by τ_{dsp} until τ_{as}.

3. DATA CORRELATION MAINTENANCE

We consider a multirate system where the shared buffer is accessed *asynchronously in a non-blocking fashion*[4]. We decrease the uncertainty in the data management usually induced by the utilization of the arbitration mechanisms (i.e semaphores, synchronisation protocols, etc.). The later may provoke unpredictable behavior such as the priority inversion problems and possible deadlock formations [2, 5, 6].

Considering that communicating tasks may sample at different rates, we split the correlating problem into 3 subproblems, namely:

PROBLEM 1. *Ensuring that all $\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ propagate the same data produced by τ_{dsp}.*

PROBLEM 2. *Setting τ_{as} to read the data resulting from the same execution step of τ_{dsp}.*

PROBLEM 3. *Computing the **optimal size** for the buffer such that Problem 1&2 have at least a solution.*

DEFINITION 1 (**Optimal size**). *A buffer size denoted by $|\beta_{dsp}|$ is optimal if it is the smallest size of β_{dsp} such that if the data read by at least one of the successors can be read by the remaining successors before being overwritten.*

3.1 Setting all successors to read the same data

We consider a set of task pairs $(\tau_{dsp}, \tau_{dsp_j}) \in E | j \in [2, k)$, where k is the number of tasks that read the data produced by τ_{dsp}. Let β_{dsp} be the buffer where τ_{dsp} instances write and the $\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ read their respective inputs. Given that we are dealing with a multirate system, it is obvious that some data samples produced by τ_{dsp} may be overwritten before being read by all the k successors or can be read several times. Thus, in order to solve the Problem 1, it is required that if a given data sample is read by at least an instance of one of the τ_{dsp_j} then this sample should not be overwritten before all the $\{\tau_{dsp}\}$ consume it.

3.1.1 Data reading management

As mentioned previously, each time an instance of the producer task completes, the *head* pointer is incremented. We denote by $lp\{\tau_{dsp_j}\}$ the task of lower priority among $\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ tasks that read the data produced by τ_{dsp}. In order to set all the *successors* to read the same data sample from β_{dsp} the following principle is imposed:

- **(i)**:$\forall\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ the value of *tail* is set by an instance of $lp\{\tau_{dsp_j}\}$ at its completion. We assume that the $lp\{\tau_{dsp_j}\}$ instances are always the last ones to read the data produced by τ_{dsp} which is responsible of pointing*where to write next time* (*head* value). Accordingly, the value of *tail* is given by the position where an instance of τ_{dsp} previously wrote; that is, $head - 1$. Formally,

$$tail_{lp\{\tau_{dsp_j}\}} \leftarrow head - 1 \quad (1)$$

- **(ii)**: Each time an instance of any of the $\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ is activated, it reads from the position pointed by *tail* pointer. We should note that the *tail* value is set by $lp\{\tau_{dsp_j}\}$ at its completion time.

$$tail_{\tau_{dsp_j}} \leftarrow tail_{lp\{\tau_{dsp_j}\}} \quad (2)$$

Further, we compute the maximum time delay that the read data can stay into the buffer before being overwritten while considering that the buffer size is optimal. We call this delay the ***Data Consistent Time Interval*** that we denote by ***DCTI*** and formally computed using the Equality 3.

3.1.2 Computing the DCTI

The Theorem 1 provides the formal way to computing the $DCTI$.

THEOREM 1. *We consider $\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ where k is the number of successors to τ_{dsp}. The DCTI is found when τ_{dsp} and $lp\{\tau_{dsp_j}\}$ are released simultaneously in such a manner that the response time of the current instance of $lp\{\tau_{dsp_j}\}$ is equal to the period of τ_{dsp} and the next instance of $lp\{\tau_{dsp_j}\}$ executes for its worst case response time. Formally,*

$$DCTI = T_{lp\{\tau_{dsp_j}\}} + R_{lp\{\tau_{dsp_j}\}} - C_{dsp} \quad (3)$$

where $T_{lp\{\tau_{dsp_j}\}}$ and $R_{lp\{\tau_{dsp_j}\}}$ are, respectively, the period and the worst case response time of $lp\{\tau_{dsp_j}\}$ and C_{dsp} the worst case execution time of τ_{dsp}.

Proof: The Equality 3 is detailed as follows

$$DCTI = \underbrace{T_{dsp} - C_{dsp}}_{(a)} + \underbrace{T_{lp\{\tau_{dsp_j}\}} - T_{dsp}}_{(b)} + \underbrace{R_{lp\{\tau_{dsp_j}\}}}_{(c)}$$

where

- (a): If τ_{dsp} and $lp\{\tau_{dsp_j}\}$ were released simultaneously and the response time of $lp\{\tau_{dsp_j}\}$ is equal to T_{dsp}, it means that by the time the data produced by τ_{dsp} is tagged (at the completion of $lp\{\tau_{dsp_j}\}$ instance), the maximum time this data will have been into the buffer is given by $T_{dsp} - C_{dsp}$.

[4]Shared variable (buffer) is accessed without any arbitration mechanism such as semaphores or any kind of synchronization protocols.

- (b): Since the execution completion of the current instant of $lp\{\tau_{dsp_j}\}$ happened at a time instant equal to T_{dsp}, the release time of the next instance of $lp\{\tau_{dsp_j}\}$ is going to happen at a time instant given by $T_{lp\{\tau_{dsp_j}\}} - T_{dsp}$.
- (c): The next data to be read is going to be tagged at the execution completion of the next instance of $lp\{\tau_{dsp_j}\}$. If the latter executes for a time equal to its worst case response time, then, the time interval between the previous and the current tagged data is the largest possible computed by the Equation 3.

3.1.3 Computing the optimal size of the buffer β_{dsp}

The maximum time delay that can separate two consecutive read data is the $DCTI$. Accordingly, the formal way to compute the optimal value of $|\beta_{dsp}|$ is given by the Theorem 2.

THEOREM 2. *We consider $\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ where k is the number of successors to τ_{dsp}. We denote by β_{dsp} the buffer where τ_{dsp} instances write. The optimal value of $|\beta_{dsp}|$ is equal to the number of τ_{dsp} instances that can be released and complete their execution within a time given by the $DCTI$ if $T_{dsp} < T_{lp\{\tau_{dsp_j}\}}$ or it is equal to 1 otherwise. Formally,*

$$|\beta_{dsp}| = \begin{cases} \left\lceil \frac{DCTI}{T_{dsp}} \right\rceil, & \text{if } T_{dsp} < T_{lp\{\tau_{dsp_j}\}}. \\ 1, & Otherwise \end{cases} \quad (4)$$

Proof: The $DCTI$ time defines the largest time that can separate two consecutive read data by the $\{\tau_{dsp_1}, \cdots, \tau_{dsp_k}\}$ where k is the number of *successors* to τ_{dsp}. The read data (tagged at the execution completion of an instance of the $lp\{\tau_{dsp_j}\}$) should not be overwritten before the next instance of $lp\{\tau_{dsp_j}\}$ completes.

In other words, there should be a sufficient buffer slots to keep the all data samples produced within the $DCTI$ time interval. Otherwise, the read data may be overwritten before the new data is set available to reading. Hence, the system of equations 4 is correct.

3.2 Maintaining data temporal correlation

The results in Section 3.1 guarantee that each data that propagate through different paths are read by all $\{\tau_{dsp_j}\}_{j=2}^{k}$ where k is the number of paths that propagate the data meant to be associated by τ_{as}. Moreover, different data paths may have different propagation delays for the data propagating from τ_{dsp} to τ_{as}. In the Sections 3.2.1 and 3.2.2 we propose a solution to this situation.

3.2.1 The correlating approach

When an instance of $lp\{\tau_{dsp_j}\}$ tags the data to be read, the dID of the tagged data sample is incremented. The tagged data is propagated trough different paths and all data related to it will have this same dID. So, when an instance of the *associator task* (τ_{as}) is activated it reads the data samples having the same dID from all the buffers where $\{\tau_{as_1}, \cdots, \tau_{as_k}\}$ output their computation results.

We assume that $lp\{\tau_{dsp_j}\}$ belongs to the path with the largest data propagation delay from τ_{dsp} to τ_{as}.

3.2.2 Setting the buffers sizes

We consider a set of k paths, $\{pth_1, \cdots, pth_k\}$, through which the data produced by τ_{dsp} propagate until τ_{as}. We denoted by The largest propagation delay it can take for the data to propagate from τ_{dsp} to τ_{as} is referred to the *worst case data propagation delay* that we denote by $WCPD_{dsp}^{as}$.

Additionally, we plan to compute, for each the $pth_i \in \Pi$, the smallest time delay it can take for the data to propagate from τ_{dsp} to τ_{as} and we denote it by $min\{delay\langle pth_i\rangle\}$. This calculation is presented here as a conjecture left as future work.

CONJECTURE 1. *We consider $(\tau_{as_i}, \tau_{as}) \in E | i \in \{2, \cdots, k\}$ where k is the number of paths trough which the data produced by τ_{dsp} propagate until τ_{as} and τ_{as_i} the last task belonging to pth_i path. Let β_{as_i} be the buffer where τ_{as_i} writes the outputs meant to be consumed by τ_{as}. Formally,*

$$|\beta_{as_i}| = \left\lceil \frac{WCPD_{dsp}^{as}}{min\{delay\langle pth_i\rangle\}} \right\rceil \quad (5)$$

4. CONCLUSION AND FUTURE WORKS

In this paper we have presented preliminary results on the consideration of both data propagation and the fulfilement of real-time constraints. Our future work includes the proof of our conjecture as well as the application of results on a drone use case study.

5. REFERENCES

[1] EmbedJournal. Implementing circular buffer in c. .

[2] T. Kloda, A. Bertout, and Y. Sorel. Latency analysis for data chains of real-time periodic tasks. In *23rd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 360–367, 2018.

[3] N. Pontisso. *Association cohérente de données dans les systèmes temps réel à base de composants: Application aux logiciels spatiaux.* PhD thesis, Institut National Polytechnique de Toulouse, 2009.

[4] N. Pontisso, P. Quéinnec, and G. Padiou. Analysis of distributed multi-periodic systems to achieve consistent data matching. *Concurrency and Computation: Practice and Experience*, (n° 2):pp. 234–249, 2013.

[5] J. Schlatow and R. Ernst. Response-time analysis for task chains in communicating threads. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 245–254, 2016.

[6] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Computers*, 1990.

[7] B. Steux. *RTMAPS, un environnement logiciel dédié à la conception d'applications embarqués tems-réel. Utilisation pour la détection automatique de véhicules par fusion radar/Vision.* PhD thesis, Ecole des mines de Paris, France, 2001.

[8] B. Steux, C. Laurgeau, L. Salesse, and D. Wautier. Fade: a vehicle detection and tracking system featuring monocular color vision and radar data fusion. In *Intelligent Vehicle Symposium, IEEE*, 2002.